Rapport du Projet

Système de Gestion des Recettes

Réalisé par :

ABDELLAOUI IBTYSSAM

Cours : JAVA ADVANCED

Année Académique : 2025-2026

Table des matières

1. Contexte Général

1.1. Introduction Détaillée

Ce projet consiste en la création d'une application de bureau, le **Recipe Management System**, développée en **JavaFX (FXML + CSS)** pour gérer un catalogue de recettes de cuisine. L'objectif est d'offrir une expérience utilisateur **locale, rapide et ergonomique** pour le CRUD (Créer, Lire, Mettre à jour, Supprimer) de recettes, incluant la gestion de leurs ingrédients et instructions. La persistance des données est assurée par une base **MySQL via JDBC**. L'architecture adoptée est une **séparation en couches** (Vue/Contrôleur, Service, DAO) pour garantir la maintenabilité et la testabilité du code.

1.2. Problématique

Le besoin principal des utilisateurs est de disposer d'un outil simple et ergonomique pour **centraliser** leurs recettes et retrouver rapidement les informations clés. Les solutions existantes et dispersées (fichiers, notes, tableurs) manquent de structuration, de cohérence et n'offrent pas de mécanisme de sauvegarde robuste.

1.3. Objectifs du Projet

Outre l'objectif fonctionnel (gestion des recettes), ce projet sert un enjeu pédagogique crucial en illustrant des concepts clés de l'ingénierie logicielle avancée :

— Mise en place d'une **architecture MVC** (Modèle-Vue-Contrôleur) propre.
— **Couche DAO** avec interfaces dédiées (`RecipeDao`, `UserDao`) et implémentations JDBC (`RecipeDaoImpl`, `UserDaoImpl`).
— Maîtrise de la **gestion transactionnelle** et de l'intégrité référentielle avec JDBC (sauvegarde atomique recette + ingrédients + instructions).
— Implémentation d'une **recherche multi-champs** (titre, description, ingrédients, instructions) avec **dédoublonnage** par identifiant.

1.4. Solution Proposée

La solution retenue est une application locale et modulaire basée sur l'architecture suivante :

— **Architecture en couches** : Vue (FXML) → Contrôleurs (JavaFX) → Services (logique métier) → DAO (JDBC) → MySQL.
— **Transactions** : Assurer l'atomicité des opérations entre l'entité parent (Recette) et ses enfants (Ingrédients, Instructions) avec commit/rollback.

— **Visibilité et Authentification** : Recettes `PUBLIC`/`PRIVATE`, propriété via `owner_user_id` ; mots de passe hachés en `BCrypt` et migration automatique des anciens mots de passe en clair.

— **Recherche** : Filtrage sur les recettes *visibles* pour l'utilisateur courant et suppression des doublons par identifiant.

— **UX Navigation** : *Liste* → *Détail* avec édition/suppression sur la page de détail (les actions ne sont plus sur la page de liste).

2. Cahier des Charges et Conception

2.1. Exigences Fonctionnelles et Non-Fonctionnelles

2.1.1. Exigences Fonctionnelles

Les fonctionnalités primaires du système comprennent :

— **CRUD** complet des Recettes.
— Gestion associée des **ingrédients** et des **instructions** pour chaque recette.
— **Recherche** par mots-clés sur le titre, la description, les ingrédients et les instructions, avec **dédoublonnage** des résultats par identifiant.
— **Filtrage** des recettes par catégorie.
— **Visibilité** des recettes (`PUBLIC`/`PRIVATE`) et **propriété** via `owner_user_id`.

2.1.2. Exigences Non-Fonctionnelles

— **Performance** : Temps de réponse acceptable pour une application de bureau locale.
— **Fiabilité** : Garantie de l'intégrité des données via les transactions JDBC et l'intégrité référentielle MySQL.
— **Maintenabilité** : Séparation stricte des couches pour un code testable et facilement maintenable (MVC, Service, DAO).
— **Sécurité** : `PreparedStatement` pour prévenir les injections SQL et **hachage BCrypt** des mots de passe avec migration automatique des comptes hérités en clair.

2.2. Outils et Technologies

Le projet repose sur la pile technologique suivante :

— **IDE** : VS Code (projet Maven).
— **Langage/SDK** : Java 17, JavaFX 17.0.2.
— **Interface Utilisateur (UI)** : JavaFX (FXML/CSS) avec la bibliothèque ControlsFX.
— **Base de données** : MySQL (MySQL Connector/J).
— **Build** : Maven (`javafx-maven-plugin`, `maven-compiler-plugin`).

2.2.1. Extrait de Configuration de Base de Données

La connexion à la base de données `recipes_db` est centralisée via la classe `DatabaseConfig`, assurant une gestion cohérente des paramètres de connexion (URL, identifiants).

```java
package com.myapp.config;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DatabaseConfig {
  private static final String URL =
    "jdbc:mysql://localhost:3306/recipes_db?useSSL=false&serverTimezone=UTC";
  private static final String USERNAME = "root";
  private static final String PASSWORD = "p@ssw0rd";

  public static Connection getConnection() throws SQLException {
    return DriverManager.getConnection(URL, USERNAME, PASSWORD);
  }
}
```

Listing 1 – `DatabaseConfig.java`

2.2.2. Modèle de Données

Le modèle relationnel est composé de quatre tables principales : `user`, `recipe`, `ingredient` et `instruction`. Les relations sont assurées par des clés étrangères :

— `recipe.owner_user_id` référence `user.id` (propriété et visibilité).

— `ingredient.recipe_id` et `instruction.recipe_id` référencent `recipe.id`.

Des index sont recommandés sur `recipe(owner_user_id)`, `recipe(visibility)` et `user(email)` pour optimiser les requêtes de filtrage et d'authentification.

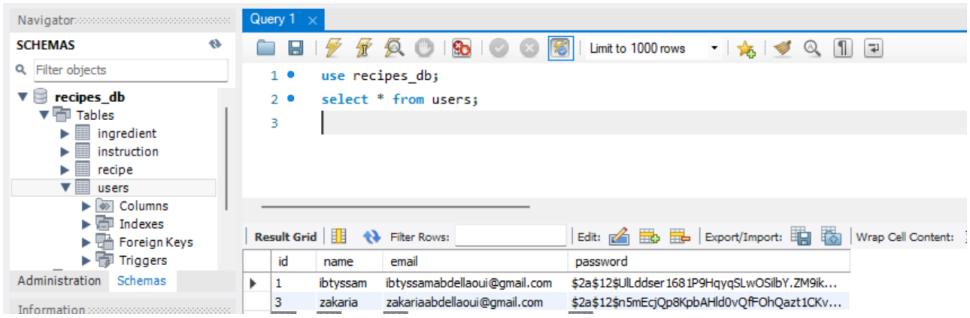

FIGURE 1 – Schéma des tables MySQL (`user`, `recipe`, `ingredient`, `instruction`)

2.2.3. Architecture en Couches (MVC, Service, DAO, JDBC)

L'application est découpée en couches strictes pour isoler les préoccupations et renforcer l'approche MVC.

— **Vue (JavaFX FXML/CSS)** : Responsable de l'affichage et de la gestion des événements de l'utilisateur (`src/main/resources/views/*.fxml`).

— **Contrôleurs (JavaFX)** : Gèrent l'interaction entre la Vue et la couche Service (`src/main/java/com/myapp`

— **Services (Logique Métier)** : Centralisent les règles de gestion et coordonnent les opérations (`RecipeServiceImpl.java`).

— **DAO (Data Access Object)** : Abstraient l'accès aux données et le **mapping** entre *Models* et MySQL. L'implémentation `RecipeDaoImpl` utilise **JDBC**.

2.2.4. Diagrammes de Conception

Diagramme de Cas d'Utilisation Le diagramme suivant montre les fonctionnalités offertes par le système à l'utilisateur.

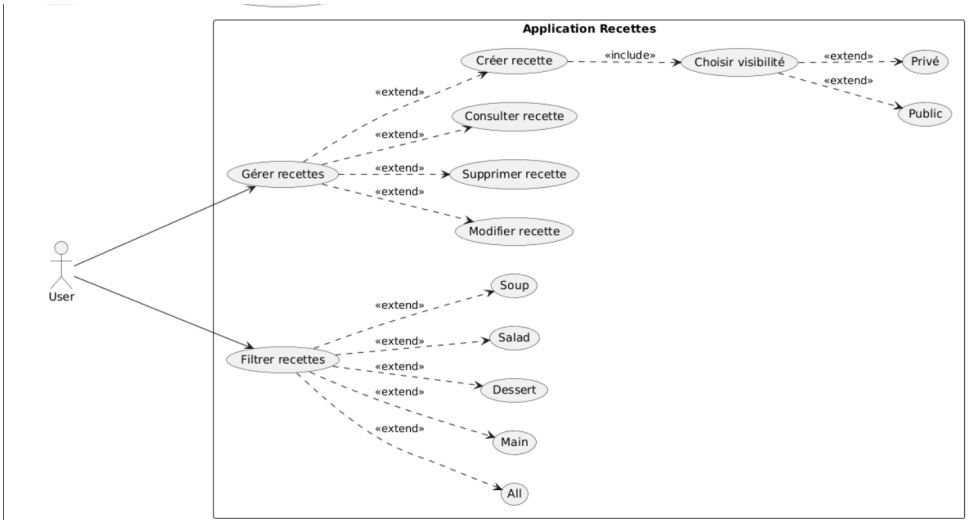

FIGURE 2 – Diagramme de cas d'utilisation

Diagramme de Séquence (Recherche) Ce diagramme illustre le flux d'exécution lors de la recherche de recettes, de la Vue à la base de données, en passant par le Controller, le Service et le DAO, avec filtrage en mémoire et **dédoublonnage** des résultats.

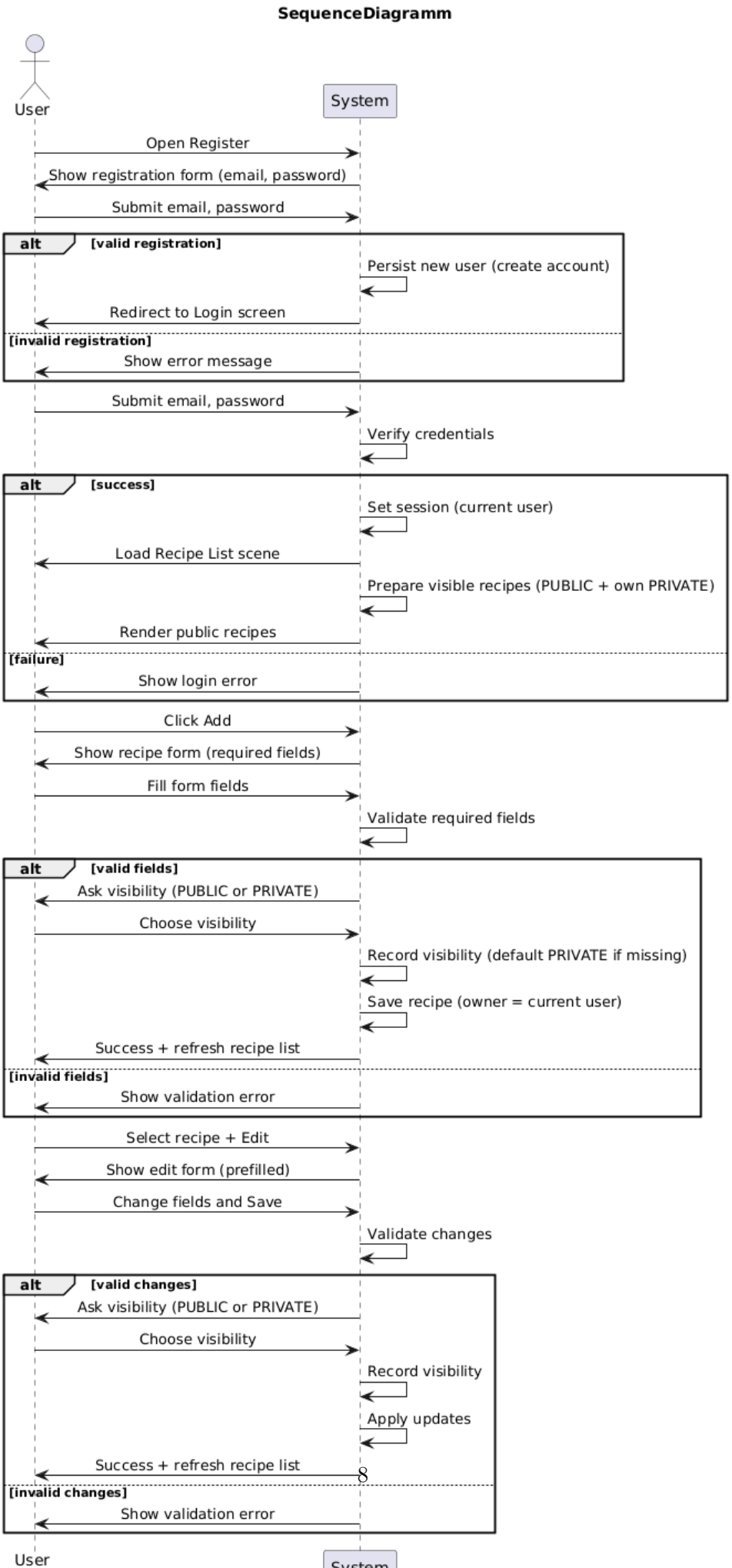
SequenceDiagramm
User
System
Open Register
Show registration form (email, password)
Submit email, password
alt
[valid registration]
Persist new user (create account)
Redirect to Login screen
[invalid registration]
Show error message
Submit email, password
Verify credentials
alt
[success]
Set session (current user)
Load Recipe List scene
Prepare visible recipes (PUBLIC + own PRIVATE)
Render public recipes
[failure]
Show login error
Click Add
Show recipe form (required fields)
Fill form fields
Validate required fields
alt
[valid fields]
Ask visibility (PUBLIC or PRIVATE)
Choose visibility
Record visibility (default PRIVATE if missing)
Save recipe (owner = current user)
Success + refresh recipe list
[invalid fields]
Show validation error
Select recipe + Edit
Show edit form (prefilled)
Change fields and Save
Validate changes
alt
[valid changes]
Ask visibility (PUBLIC or PRIVATE)
Choose visibility
Record visibility
Apply updates
Success + refresh recipe list
[invalid changes]
Show validation error
User
System

Diagramme de Classe Le diagramme de classe ci-dessous présente les principales entités et les dépendances entre les couches (Contrôleurs, Services, DAO).

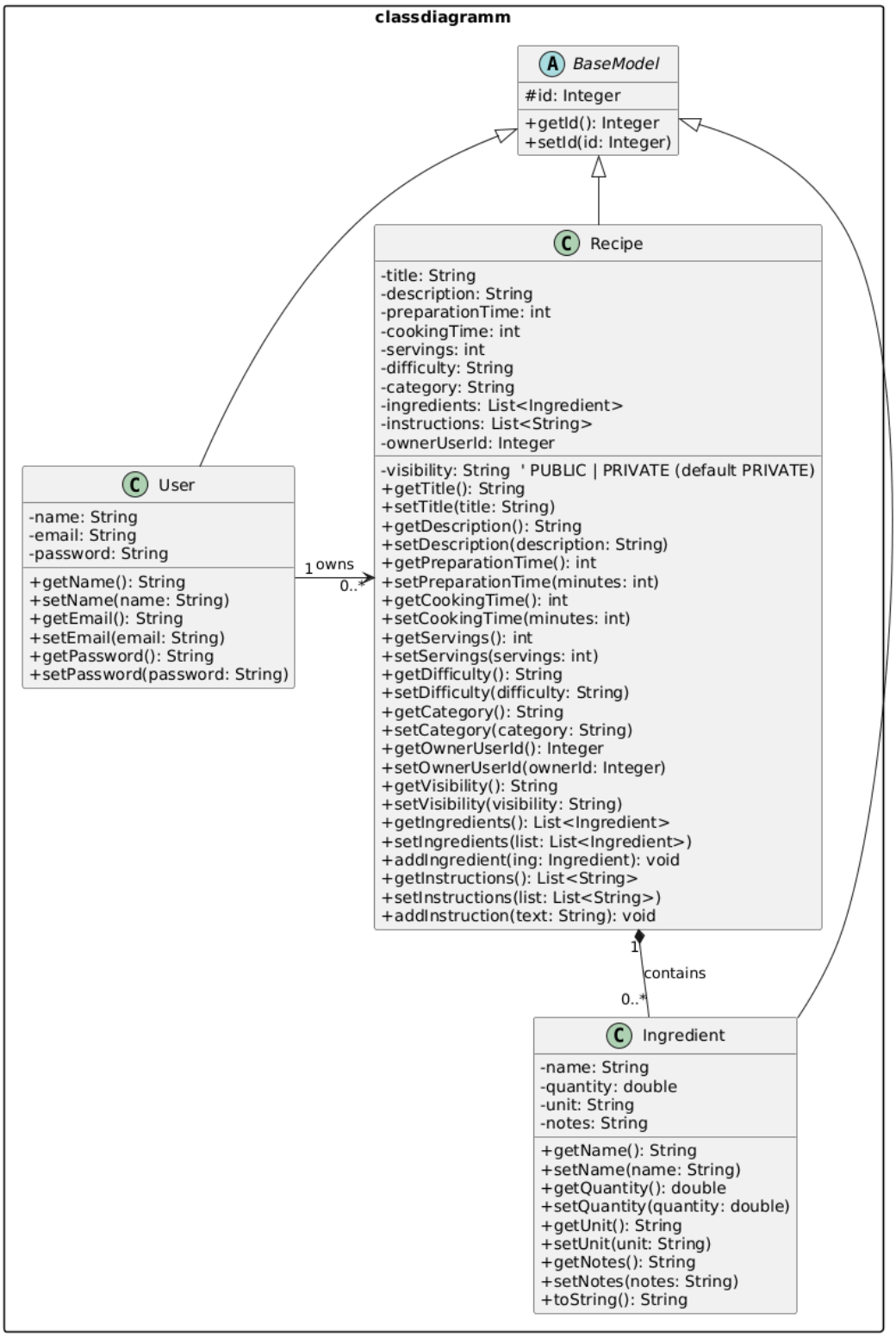

FIGURE 4 – Diagramme de classe (Controllers → Services → DAO → DatabaseConfig ; entités `User`, `Recipe`, `Ingredient`)

3. Réalisation et Implémentation

3.1. Architecture Implémentée (Vue d'ensemble)

L'implémentation concrétise la séparation des couches :

— La classe `Main.java` initialise l'interface JavaFX (charge `home.fxml`).

— `RecipeServiceImpl` gère la logique métier et délègue l'accès aux données à `RecipeDaoImpl` via l'interface `RecipeDao`.

— `RecipeDaoImpl` est responsable du CRUD, des transactions et du mapping des entités (*recette, ingrédients, instructions*) avec **JDBC**.

— `AuthServiceImpl` gère l'authentification (hachage **BCrypt**) et `SessionManager` stocke l'utilisateur courant pour filtrer les recettes `PUBLIC/PRIVATE`.

3.2. Concepts Clés et Extraits de Code

3.2.1. Polymorphisme avec des Interfaces DAO

Les interfaces `RecipeDao` et `UserDao` définissent le contrat standard de persistance, illustrant le **polymorphisme** par implémentation (`RecipeDaoImpl`, `UserDaoImpl`).

```
package com.myapp.dao;

import java.util.List;
import java.util.Optional;
import com.myapp.models.Recipe;

public interface RecipeDao {
  Recipe save(Recipe recipe);
  Optional<Recipe> findById(Integer id);
  List<Recipe> findAll();
  void delete(Integer id);
  List<Recipe> findAllVisibleForUser(Integer userId);
}
```

Listing 2 – `RecipeDao.java` (Contrat DAO)

```
package com.myapp.dao;

import java.util.Optional;
import com.myapp.models.User;

public interface UserDao {
  Optional<User> findByEmail(String email);
  User save(User user);
  void updatePasswordHash(int userId, String hashedPassword);
```

```
10 }
```

Listing 3 – `UserDao.java` (Contrat DAO)

3.3. Interface Utilisateur et Captures d'Écran

Cette section présente le rendu graphique de l'application, illustrant l'ergonomie de l'interface utilisateur développée en JavaFX.

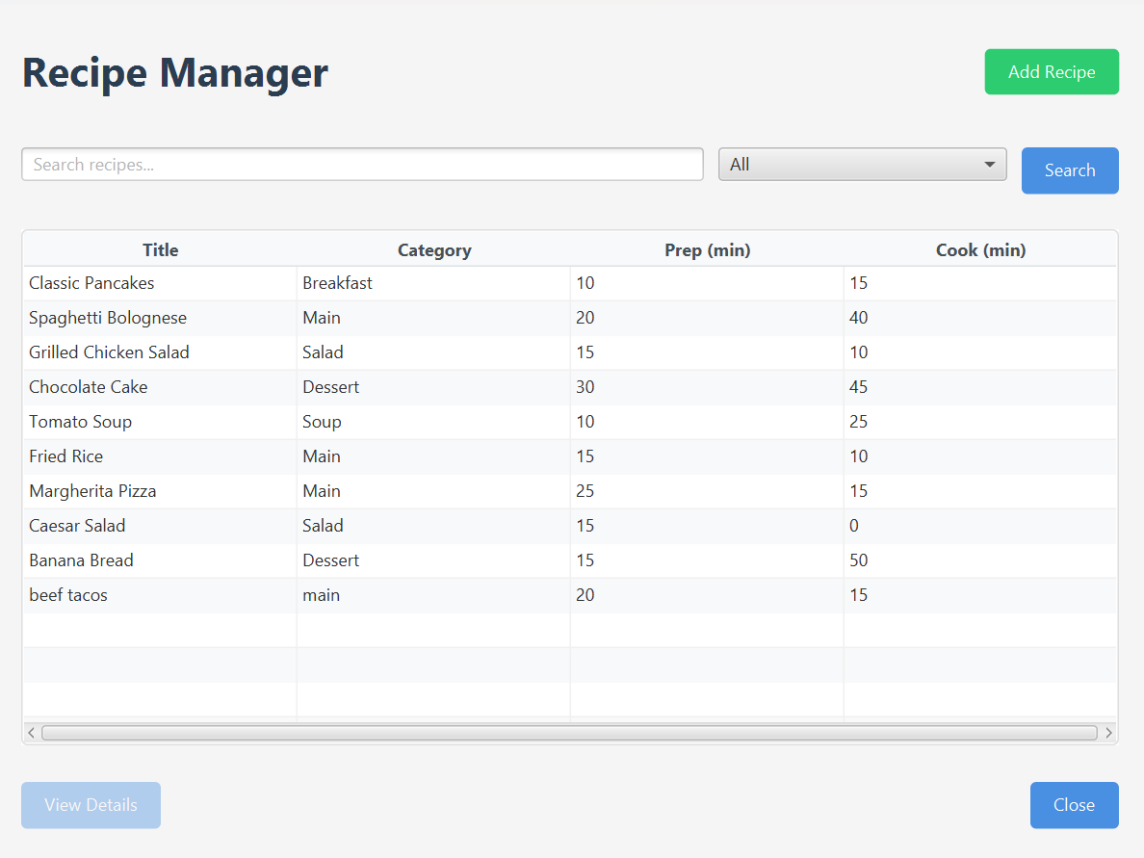

FIGURE 5 – Écran de la Liste des Recettes et Fonctionnalité de Recherche

La première vue (Figure 5) permet la consultation du catalogue et l'accès à la recherche.

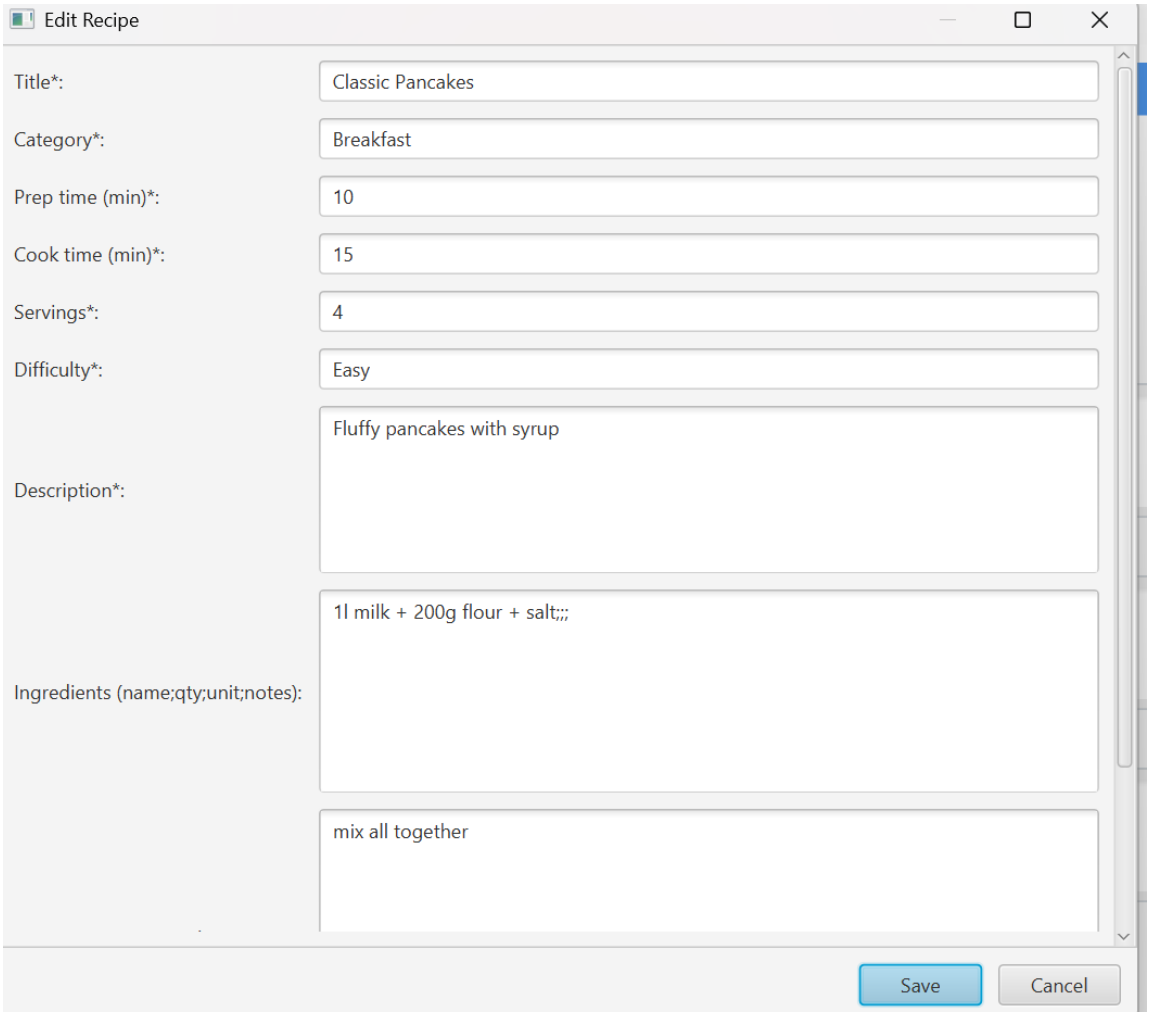

FIGURE 6 – Écran de Détail et de Modification d'une Recette

La Figure 6 illustre le formulaire de modification sur la page.

3.4. Structure du Projet

La structure du projet reflète la séparation stricte des couches (`config`, `controllers`, `dao`, `models`, `services`) et l'utilisation de Maven.

```
javaadvanced/
 pom.xml
 setup_tables.sql
 src/
   main/
     java/
       module-info.java
       com/myapp/
         Main.java
```

```
        config/DatabaseConfig.java
        controllers/
          HomeController.java
          LoginController.java
          RegisterController.java
          RecipeListController.java
          RecipeDetailController.java
        dao/
          RecipeDao.java
          RecipeDaoImpl.java
          UserDao.java
          UserDaoImpl.java
        models/
          BaseModel.java
          Ingredient.java
          Recipe.java
          User.java
        services/
          AuthService.java
          AuthServiceImpl.java
          RecipeService.java
          RecipeServiceImpl.java
          SessionManager.java
    resources/
      styles/styles.css
      views/
        home.fxml
        login.fxml
        register.fxml
        recipe-list.fxml
        recipe-detail.fxml
target/ (généré par Maven)
```

4. Conclusion et Perspectives

4.1. Bilan des Objectifs Atteints

Ce projet de **Système de Gestion des Recettes** développé en JavaFX et MySQL a atteint les objectifs fixés, tant sur le plan fonctionnel que pédagogique.

Sur le plan fonctionnel, l'application fournit un outil de bureau fiable pour le cycle de vie complet des recettes (CRUD), incluant la gestion détaillée des **ingrédients** et des **instructions**. La navigation *Liste* → *Détail* centralise l'édition/suppression sur la page de **détail**. La **recherche multi-champs** opère sur les recettes *visibles* (PUBLIC/PRIVATE via `owner_user_id`) et applique un **dédoublonnage** par identifiant pour éviter les doublons.

Sur le plan pédagogique, le projet a validé les concepts avancés du cours *JAVA ADVANCED* :

— Architecture en couches stricte (MVC, Service, DAO) pour la maintenabilité et la séparation des préoccupations.
— Accès aux données via **JDBC**, avec **transactions** explicites au niveau DAO (sauvegarde atomique recette + enfants).
— **Sécurité applicative** : usage systématique de `PreparedStatement` contre l'injection SQL et **hachage BCrypt** des mots de passe (avec migration automatique des comptes hérités).
— **Polymorphisme** au travers des **interfaces DAO** (`RecipeDao`, `UserDao`) et de leurs implémentations (`RecipeDaoImpl`, `UserDaoImpl`).

4.2. Forces et Faiblesses du Projet

4.2.1. Forces

Les principaux atouts du projet résident dans la clarté de son architecture, le contrôle précis des **transactions** au niveau DAO, et l'ergonomie de l'interface JavaFX (édition sur la page de détail). La sécurité basique est renforcée par `PreparedStatement` et le **BCrypt** pour les mots de passe. Le **filtrage de visibilité** (PUBLIC/PRIVATE) et le **dédoublonnage** des résultats de recherche améliorent la cohérence des données affichées.

4.2.2. Axes d'Amélioration

Plusieurs pistes d'amélioration sont envisageables :

— **ORM** : Étudier une migration vers *JPA/Hibernate* pour simplifier le mapping et la gestion des transactions.
— **Tests** : Ajouter des tests unitaires (Service) et d'intégration (DAO) pour garantir la non-régression.

5. Compléments de Contenu

5.1. Filtrage par Session

Le service opère uniquement sur les recettes *visibles* pour l'utilisateur courant. `SessionManager` fournit l'`userId` à `RecipeServiceImpl`, qui interroge `RecipeDao.findAllVisibleForUser(userId)` (recettes `PUBLIC` + recettes dont `owner_user_id` = utilisateur).

5.2. Comportement de la Recherche et Dédoublonnage

Pour les requêtes **courtes** (longueur ≤ 2), la recherche privilégie le **titre** afin de limiter le bruit. Pour les requêtes **plus longues**, le filtrage s'étend à la **description**, aux **ingrédients** et aux **instructions**. Les résultats sont **dédoublonnés** par `id` (première occurrence conservée), puis les éléments sans `id` sont ajoutés en fin de liste pour garantir l'absence de doublons à l'écran.

5.3. Module BCrypt

L'implémentation du hachage des mots de passe repose sur **BCrypt**. Le module Java est configuré avec `requires jbcrypt;` dans `module-info.java` afin de rendre la bibliothèque accessible au runtime.

5.4. Migration du Schéma

L'introduction de `owner_user_id` et `visibility` dans la table `recipe`, ainsi que la table `user`, est réalisée via la migration `migrations/20251228_add_auth.sql`. Cette migration ajoute les **FKs**, les **index** (`user(email)`, `recipe(owner_user_id)`, `recipe(visibility)`) et les contraintes nécessaires.

5.5. Validation et Gestion des Erreurs

Les contrôleurs effectuent des validations sur les champs obligatoires et les valeurs numériques (**préparation**, **cuisson**, **portions**). En cas d'entrée invalide ou d'erreur de persistance, une **alerte JavaFX** explicite informe l'utilisateur (ex. : nombres attendus pour les durées/portions).